TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Intelligente Netze und Management verteilter Systeme
Prof. Anja Feldmann, Ph. D.
Harald Schiöberg, Bernhard Ager, Vlad Manilici

12th Assignment Protocol Design WS 08/09

Question 1: (40 points) Implementation of a Simple Forwarding Table

As flooding is a very inefficient way for forwarding messages, we add a simple mechanism for optimizing message forwarding. This mechanism consists of a simple lookup table. The entries of this table a generated using received messages and allow us to forward messages for known destinations in the 'correct' direction. The table is called a *forwarding table*.

The forwarding table has to constructed and used in the following way (learning phase):

- Extract the node ID of the originator (`FROM`) and the TTLs of all (non-handshake) messages. Then insert a new entry or, if the TTL hints at a better (shorter) path, update an existing entry in the forwarding table.

  The node ID of the originator is to be used as key to access the appropriate entry in the table. Thus, a table entry looks like this:

  Timestamp: when has this entry been created or updated last

  Direction: the neighbour (neighbour connection) the message was received from

  TTL: the TTL of the message used when adding/changing this entry. The TTL is a measure for the distance to the node, as it is always initialized with 3.

  Using this mechanism we get a table with the following property: For each node from which we have received a message, the table contains the neighbour that the originitaor used to reach us. This also means that we can reach the originator via the same neighbour!
- If a neighbour connection dies, all table entries that refer to this neighbour have to be removed. This might lead to nodes not reachable anymore through the forwarding table.

Using this forwarding table, it is now possible to optimize the forwarding of messages:

- Table entries older than 120 seconds must be ignored and removed.
- Messages with a specific destination (`FOR`) for which we have a current table entry are no longer flooded. Instead we forward the message only over the corresponding neighbour connection found in the forwarding table.
- If there is no matching entry for the destination node in the forwarding table, the message has to be flooded as before.
- No matter if a messages is forwarded by flooding or using the forwarding table, the TTL of the message has to be decremented and no forwarding is allowed if the decremented TTL is 0.
- The table logic should be implemented so that it is possible to manage (add, update, remove, lookup) entries using dedicated function calls.

Deliverables:

- Your program/script (both source code and binary if written in C, C++ or Java)
- A short description of how to use the software, and what things are not quite working yet.

Question 2: (20 points) Automated Session Establishment

For this task, the nodes learned during the session establishment phase (HELLO handshake) of the protocol are to be used to enable the node software to automatically setup sessions to new neighbours. Therefore the following mechanisms have to be implemented:

- Store all peer node IDs learned via the `NEIGHBOURS` part of the `HELLO` handshake during the session initialization in a queue (FIFO).
- As long as there are less than the maximum number of 4 *actively established* sessions, new sessions have to be established. The peer nodes to connect are taken from the queue, i.e., removed from the queue and then used.
- If a session is closed or dies, the node ID of the neighbour has to be re-added to the queue. In addition, a new session has to be established, as long as there are less than 4 *actively established* sessions.
- If the attempt to connect a node fails, the node-ID must be ignored and is removed from the queue. Then the software has to attempt to connect to the next node from the queue.

Deliverables:

- Your program/script (both source code and binary if written in C, C++ or Java)
- A short description of how to use the software, and what things are not quite working yet.

Question 3: (40 points) File transfer in the P2P network

The simplest and most efficient method to transfer a file from one P2P node to the other is via a direct connection between the two nodes concerned. A download is initiated with a `GET`-Message (see assignment sheets 10/11). Such a `GET` request is specified as follows (but, as hinted by the arrow, within a single line):

```
GET FROM <node id> FOR <node id> KEY <filename> ↲
                    MESSAGE-ID <id> TTL <ttl> P2P/0.1\r\n
```

The `FROM`-field specifies the node that wants to download the file. The `FOR`-field specifies a node of which we know that it has a copy of the requested file (cf. `SEARCH` and `FOUND`). When the `FOR` node receives such a `GET`-message it creates a new TCP listening socket and publishes the listening port in a response message of type `DIRECTCONNECT`. The reply code is 380. The response messages have the following format (but, as hinted by the arrow, within a single line):

```
P2P/0.1 380 DIRECTCONNECT FOR <node id> FROM <node id> MESSAGE-ID <id> ↲
                    KEY <filename> SIZE <size> PORT <port> TTL <ttl>\r\n
```

The `FOR`, `FROM`, and `TTL` fields are used as usual for specifying sender, receiver, and the TTL of the message. The `KEY`-field in the first line is always copied from the `GET` message, but note that every message has a new, unique `MESSAGE-ID`. After receiving a `DIRECTCONNECT` message the receiver opens a TCP connection to the sender with the given `PORT`, and the sender will send the file without any further protocol. The `SIZE` field is used by the receiver to check if the download succeded. In addition to that, specify and implement how `DIRECTCONNECT` could be used for `PUT` requests. How should the `SIZE` field be used?

Deliverables:

- Your program/script (both source code and binary if written in C, C++ or Java)
- As text file: How did you use the `DIRECTCONNECT` fields for `PUT` requests?

Due Date: Wednesday, 04.02.2009 at 23:59 s.t.